REMARKS/ARGUMENTS

Claims 1-5 and 21-35 are pending in the present application. Claims 6-20 are canceled. Claims 1-5 and 21-22 are amended. Claims 23-35 are added. Support for the amendments to claims 1-5 and 21-22 and new claims 23-35 is located at least in previous drafts of the claims and in the Specification on page 2, line 10, through page 3, line 32; on page 4, lines 4-25; on page 5, lines 17-32; on page 6, line 37, through page 9, line 14; and in Figures 1 and 3. Reconsideration of the claims is respectfully requested.

I. 35 U.S.C. § 102, Anticipation

The Examiner has rejected claims 1, 2, 5, 14, 20 and 21 under 35 U.S.C. § 102 as being anticipated by Fresko et al., U.S. Patent No. 5,966,702 (hereinafter "*Fresko*"). This rejection is respectfully traversed.

The Examiner states:

> Regarding Claims 1, Fresko teaches: A data processing method for creating an executable file by combining a plurality of run units, the method comprising:
>
> identifying a first data entity and a second data entity, wherein both the first data entity and the second data entity are identified using an Assembler instruction, and wherein the Assembler instruction identifies character strings which are required to appear only once in the executable file; **(Column 9, Lines 17-21, "In step 402, the preprocessor examines the constant pool tables of each class to determine the set of class file constants (such as strings and numerics, as well as others specific to the class file format) that can be shared between classes in "S."" i.e. determines which entities are "constants" see also Col. 8, Ln 10-14"Constant pool table 305 is a table of variable-length data structures representing various string constants, numerical constants, class names, field names, and other constants that are referred to within the ClassFile structure." See further Col. 9, Ln 3-5"Also, by implementing a shared constant table, entries in the constant table need be fully resolved at most once."))**
>
> responsive to a determination that a first run unit to be added to the executable file comprises a first data entity set to a first value indicating that the first data entity is required to appear only once in the executable file **(Column 9, Lines 17-21, "In step 402, the pre-processor examines the constant pool tables of each class to determine the set of class file constants (such as strings and numerics, as well as others specific to the class file format) that can be shared between classes in "S."" i.e. determines which entities are**

"constants"), determining whether the first data entity matches a second data entity set to a second value and included in a second run unit, wherein the second run unit comprises a run unit that was previously added to the executable file **(Column 9, Lines 21-23, "A shared constant pool table is created in step 403, with all duplicate constants determined from step 402.")** *[here the examination of classes inherently is carried out sequentially, and thus the first class examined/inserted in the class file reads on "a run unit that was added previously"]*;

responsive to a determination that the first data entity matches the second data entity, adding the first run unit to the executable file [[but]] without the first data entity **(Column 9, Lines 23-35, "In step 404, the pre-processor removes the duplicate, shared constants from the individual constant pool tables of each class.").** ; and responsive to a determination that the first data entity does not match the second data entity, adding the first run unit to the executable file with the first data entity **(Column 9, Lines 23-35, "In step 404, the pre-processor removes the duplicate, shared constants from the individual constant pool tables of each class." i.e. if they are NOT duplicates, they are NOT removed from the constant pool of the class when the class file is added).**

Office Action dated February 24, 2009, pages 2-4.

A prior art reference anticipates the claimed invention under 35 U.S.C. § 102 only if every element of a claimed invention is identically shown in that single reference, arranged as they are in the claims. *In re Bond*, 910 F.2d 831, 832, 15 U.S.P.Q.2d 1566, 1567 (Fed. Cir. 1990). All limitations of the claimed invention must be considered when determining patentability. *In re Lowry*, 32 F.3d 1579, 1582, 32 U.S.P.Q.2d 1031, 1034 (Fed. Cir. 1994). Anticipation focuses on whether a claim reads on the product or process a prior art reference discloses, not on what the reference broadly teaches. *Kalman v. Kimberly-Clark Corp.,* 713 F.2d 760, 218 U.S.P.Q. 781 (Fed. Cir. 1983). In this case each and every feature of the presently claimed invention is not identically shown in *Fresko*, arranged as they are in the claims, and, accordingly, *Fresko* does not anticipate the claims. Specifically, *Fresko* does not teach or suggest that "at least two of the plurality of Assembler modules include at least one DL Assembler instruction," and "identifying, by a linkage editor, a first data entity in a first run unit of the plurality of run units using a first DL Assembler instruction, and a second data entity in a second run unit of the plurality of run units using a second DL Assembler instruction, wherein the first DL Assembler instruction and the second DL Assembler instruction identify character strings which are required to appear only once in the executable file, further comprising: setting the first data entity to a first value included in the first DL Assembler instruction, and setting the second data entity to a second value included in the second DL Assembler instruction," as recited in amended independent claim 1.

Fresko is directed to a method and apparatus for pre-processing and packaging class files. Embodiments remove duplicate information elements from a set of class files to reduce the size of individual class files and to prevent redundant resolution of the information elements. Fresko's method is different that the claims of Applicants' patent application for identifying and eliminating duplicate data entities. The Office Action refers to the following portions of *Fresko* in the rejection of independent claim 1:

> In **FIG. 1**, server **100** comprises Java development environment **104** for use in creating the Java class files for a given application. The Java development environment **104** provides a mechanism, such as an editor and an applet viewer, for generating class files and previewing applets. A set of Java core classes **103** comprise a library of Java classes that can be referenced by source files containing other/new Java classes. From Java development environment **104**, one or more Java source files **105** are generated. Java source files **105** contain the programmer readable class definitions, including data structures, method implementations and references to other classes. Java source files **105** are provided to Java compiler **106**, which compiles Java source files **105** into compiled ".class" files **107** that contain bytecodes executable by a Java virtual machine. Bytecode class files **107** are stored (e.g., in temporary or permanent storage) on server **100**, and are available for download over network **101**.

Fresko, column 3, lines 18-35.

> The constant pool count value **304** identifies the number of entries in constant pool table **305**. Constant pool table **305** is a table of variable-length data structures representing various string constants, numerical constants, class names, field names, and other constants that are referred to within the ClassFile structure. Each entry in the constant pool table has the following general structure:

```
cp_info   {
     u1 tag;
     u1 info[ ];
}
```

> where the one-byte "tag" specifies a particular constant type. The format of the info[ ] array differs based on the constant type. The info[ ] array may be a numerical value such as for integer and float constants, a string value for a string constant, or an index to another entry of a different constant type in the constant pool table. Further details on the constant pool table structure and constant types are available in Chapter 4 of Section A.

Fresko, column 8, lines 9-29.

> Embodiments of the invention examine the constant pool table for each class in a set of classes to determine where duplicate information exists. For example, where two or more classes use the same string constant, the string constant may be removed from each class file structure and placed in a shared constant pool table. In the simple case, if N classes have the same constant entry, N units of memory space are taken up in storage resources. By removing all constant entries and providing one shared entry, N-1 units of memory space are freed. The memory savings increase with N. Also, by implementing a shared constant table, entries in the constant table need be fully resolved at most once. After the initial resolution, future code references to the constant may directly use the constant.

Fresko, column 8, line 61, through column 9, line 9.

> The method begins in step **400** with a set of arbitrary class files "S" (typically part of one application). In step **401**, the pre-processor reads and parses each class in "S." In step **402**, the pre-processor examines the constant pool tables of each class to determine the set of class file constants (such as strings and numerics, as well as others specific to the class file format) that can be shared between classes in "S." A shared constant pool table is created in step **403**, with all duplicate constants determined from step **402**. In step **404**, the pre-processor removes the duplicate, shared constants from the individual constant pool tables of each class.
>
> In step **405**, the pre-processor computes the in-core memory requirements of each class in "S," as would normally be determined by the class loader for the given virtual machine. This is the amount of memory the virtual machine would allocate for each class, if it were to load each class separately. After considering all classes in "S" and the additional memory requirement for the shared constant pool table, the total memory requirement for loading "S" is computed in step **406**.

Fresko, column 9, lines 15-35.

Fresko discloses that the constant pool table for each class in a set of classes is parsed and examined to determine where duplicate constants exist. All constants in each constant pool table are checked. All duplicate constants are removed from each class file and placed in the shared constant pool table. This is different that using a DL Assembler instruction to identify specific character strings which are required to appear only once in the executable file. Other Assembler instructions include character strings that are not identified. Further, there is no need in *Fresko* to mark constants with an instruction that identifies a specific character string which is required to appear only once in the executable file since all constants are checked for duplicates. *Fresko* does not teach or suggest that "at least two of the plurality of Assembler modules include at least one DL Assembler instruction," as recited in amended independent claim 1. In addition, *Fresko* does not teach or suggest "identifying, by a linkage editor, a first data entity in a first run unit of the plurality of run units using a first DL Assembler instruction, and a second data entity in a second run unit of the plurality of run units using a second DL Assembler instruction, wherein the first DL Assembler instruction and the second DL Assembler instruction identify character strings which are required to appear only once in the executable file, further comprising: setting the first data entity to a first value included in the first DL Assembler instruction, and setting the second data entity to a second value included in the second DL Assembler instruction," as recited in amended independent claim 1.

In view of the above, Applicants respectfully submit that *Fresko* does not teach each and every feature of amended independent claim 1, as is required under 35 U.S.C § 102. In addition, *Fresko* does not teach each and every feature of dependent claims 2, 5, and 22 at least by virtue of their dependency on claim 1. Claims 14 and 21 are canceled. Accordingly, Applicants respectfully request withdrawal of the rejection of claims 1, 2, 5, 14, and 21-22 under 35 U.S.C § 102.

In addition to being dependent on independent claim 1, claims 5 and 22 also distinguish over the *Fresko* reference based on the specific features recited therein. With respect to claim 5, *Fresko* does not teach or suggest that "identifying a first data entity in a first run unit of the plurality of run units using a first DL Assembler instruction, further comprises: locating data entity records from the plurality of records in the first run unit using each key value of the plurality of records in the first run unit to identify a DL type of Assembler instruction by which each of a plurality of data entities is marked, wherein each record of the plurality of records in the first run unit with a key value that identifies the DL type of Assembler instruction is identified as a data entity record in the plurality of data entity records, and wherein each record of the plurality of records in the first run unit with a key value that does not identify the DL type of Assembler instruction is not identified as a data entity record in the plurality of data entity records; and creating the first data entity by combining the plurality of data entities." As claimed, the DL Assembler instruction marks a data entity, such as a specific character string constant, that needs to appear only once in the executable file. Character string constants that are included in records of a run unit and are not associated with a DL Assembler instruction are not located, and are added to the executable without checking for duplicates. Therefore, some character string constants may be duplicates. In *Fresko*, a DL instruction is not needed since all duplicate constants are moved to a shared constant table. *Fresko* discloses a different method for removing duplicate constants that also produces different results.

In addition, *Fresko* and *Ziedman*, taken alone or in combination, do not teach or suggest that "the DL type of Assembler instruction denotes a non-executable data entity which needs only be included once in the executable file," as recited in claim 22. Applicants respectfully disagree that a shared constant table anticipates a DL type of Assembler instruction. A DL Assembler instruction marks a specific character string which needs only be included once in the executable file. Character strings are also part of other Assembler instructions. *Fresko* does not identify which character strings are not to be duplicated. To the contrary, *Fresko* discloses removing all duplicate constant entries and providing one shared entry.

II. 35 U.S.C. § 103, Obviousness

The Examiner has rejected claims 3 and 4 under 35 U.S.C. § 103 as being unpatentable over Fresko in view of Ziedman, U.S. Patent Application Publication No. 2005/0114840 (hereinafter "Ziedman"). This rejection is respectfully traversed.

The Examiner bears the burden of establishing a *prima facie* case of obviousness based on prior art when rejecting claims under 35 U.S.C. § 103. *In re Fritch*, 972 F.2d 1260, 23 U.S.P.Q.2d 1780 (Fed. Cir. 1992). The prior art reference (or references when combined) must teach or suggest all the claim limitations. *In re Royka,* 490 F.2d 981, 180 USPQ 580 (CCPA 1974). In determining obviousness, the scope and content of the prior art are... determined; differences between the prior art and the claims at issue are... ascertained; and the level of ordinary skill in the pertinent art resolved. Against this background the obviousness or non-obviousness of the subject matter is determined. *Graham v. John Deere Co.*, 383 U.S. 1 (1966). "Often, it will be necessary for a court to look to interrelated teachings of multiple patents; the effects of demands known to the design community or present in the marketplace; and the background knowledge possessed by a person having ordinary skill in the art, all in order to determine whether there was an apparent reason to combine the known elements in the fashion claimed by the patent at issue." *KSR Int'l. Co. v. Teleflex, Inc.*, No. 04-1350 (U.S. Apr. 30, 2007). *"Rejections on obviousness grounds cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness.* Id. (citing *In re Kahn*, 441 F.3d 977, 988 (CA Fed. 2006))."

Since claims 3 and 4 depend from independent claim 1, the same distinctions between *Fresko* and the invention recited in claim 1 apply to dependent claims 3 and 4. In addition, *Ziedman* does not provide for the deficiencies of *Fresko* with regard to amended independent claim 1. *Ziedman* is directed to a software tool for detecting plagiarism in computer source code. *Ziedman* is cited for teaching a "partial word matching" algorithm. *Ziedman* does not teach or suggest "," as recited in amended independent claim 1. Thus, any alleged combination of *Fresko* with *Ziedman* still would not result in the invention recited in amended independent claim 1 from which claims 3 and 4 depend. Accordingly, Applicants respectfully request withdrawal of the rejection of claims 3 and 4 under 35 U.S.C. § 103.

In addition, *Fresko* and *Ziedman*, taken alone or in combination, do not teach or suggest "identifying a third data entity using a third DL Assembler instruction that identifies character strings which are required to appear only once in the executable file, wherein the third data entity is set to a third value included in the third DL Assembler instruction," and that "responsive to a determination that the first data entity partially matches the third data entity, removing the first data entity from the executable file," as recited in amended claim 4. *Fresko* is cited for allegedly teaching the features of claim 4. As discussed above, *Fresko* does not teach or suggest a DL Assembler instruction. In addition, *Fresko* does not teach or suggest removing constants that partially match. Further, this type of feature would enter errors in *Fresko* since changing the value of a constant may cause an application to function differently than intended or even to crash.

III. <u>New Claims 23-35</u>

Claims 23-35 are computer program product and apparatus claims that contain the same subject matter of the method claims 1-5 and 21-22. Therefore, *Fresko* and *Ziedman*, taken alone or in combination, do not teach or suggest the features of claims 23-35 for the same reasons discussed above.

IV. Conclusion

It is respectfully urged that the subject application is patentable over the cited references and is now in condition for allowance.

The Examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the Examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

DATE: May 22, 2009

Respectfully submitted,

GHG/VJA

/Gerald H. Glanzman/

Gerald H. Glanzman
Reg. No. 25,035
Yee & Associates, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 385-8777
Attorney for Applicants